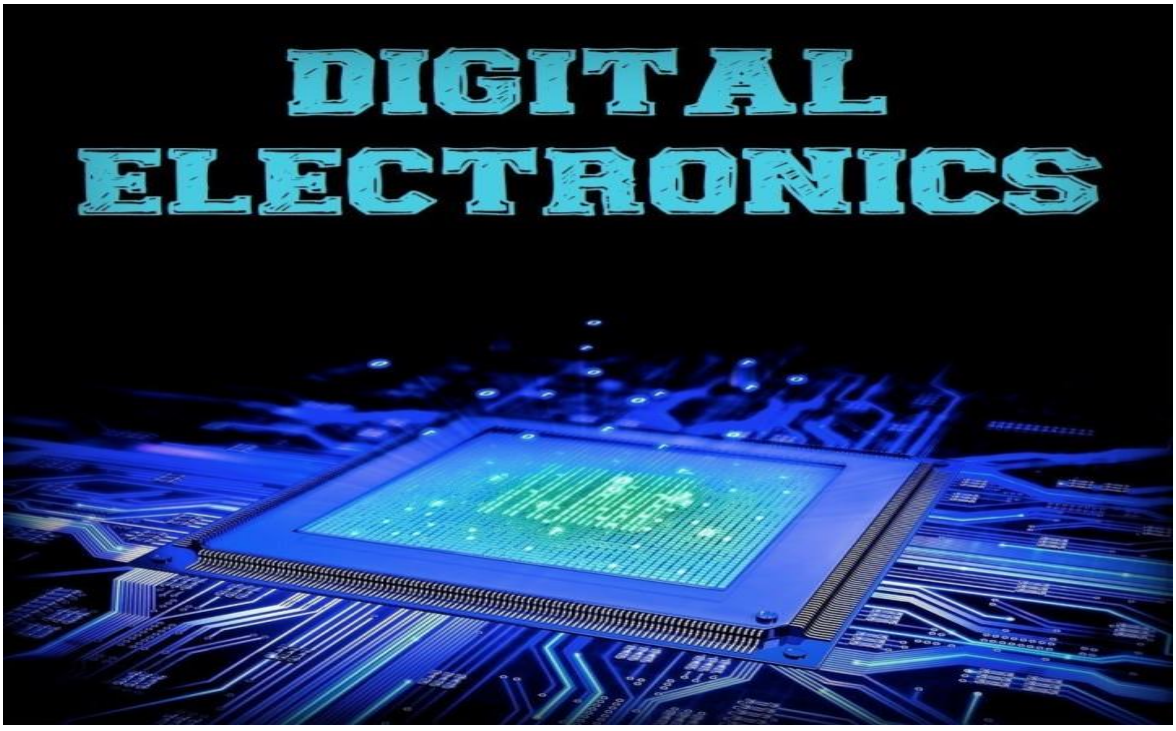
Abanob Evram

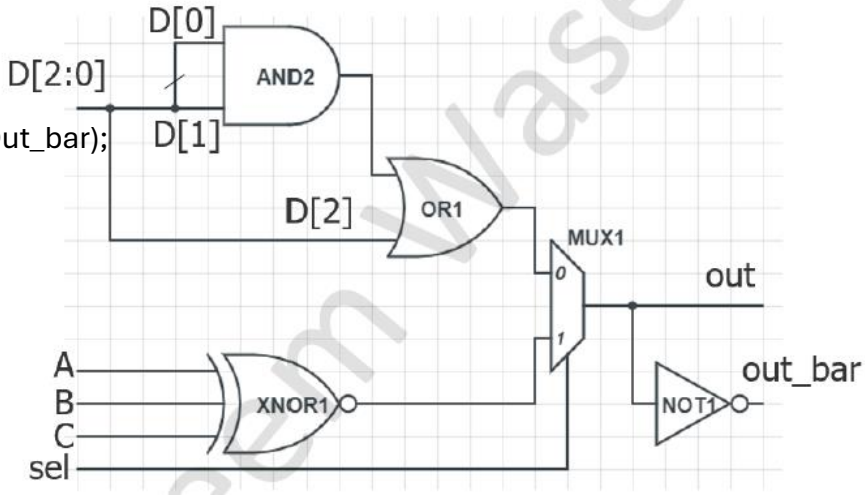
Assignmen2



- The design has 5 inputs and 2 outputs
- Randomize the stimulus in the testbench and make sure that the output is correct from the waveform

[Q1]

The design code:

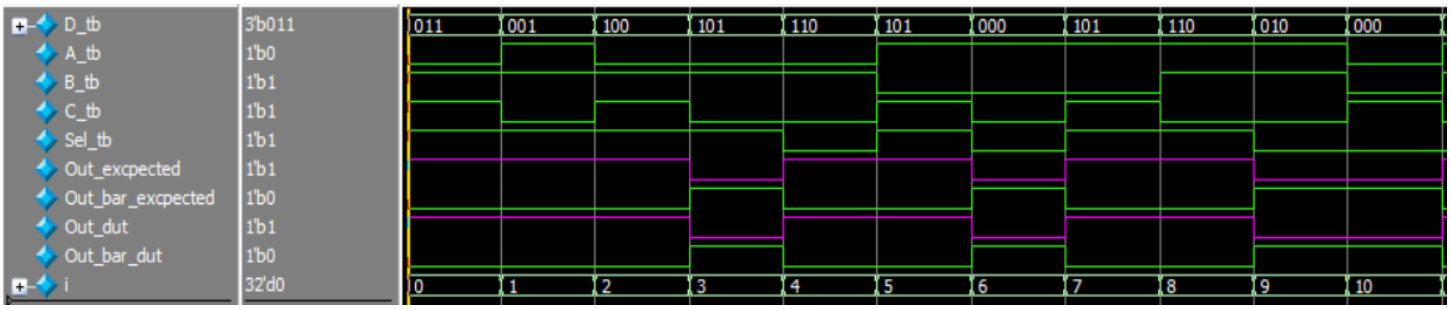


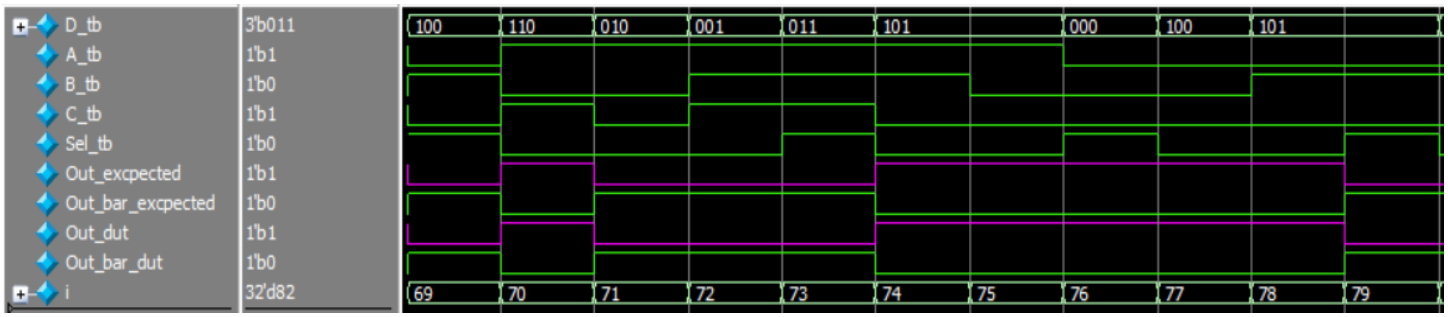

```
module Q1(D,A,B,C,Sel,Out,Out_bar);
input [2:0] D;
input A,B,C,Sel ;
output Out,Out_bar;
wire Z0,Z1,Z2;
assign Z0=D[0]&D[1];
assign Z1=Z0|D[2];
assign Z2=~(A^B^C);
assign Out=(Sel==1)?Z2:Z1;
assign Out_bar=~(Out);
endmodule
```

The testbench code:

```
module Q1_tb_randomized();
reg [2:0] D_tb;
reg A_tb,B_tb,C_tb,Sel_tb,Out_excpected,Out_bar_excpected;
wire Out_dut,Out_bar_dut;
Q1 Dut(D_tb,A_tb,B_tb,C_tb,Sel_tb,Out_dut,Out_bar_dut);
integer i ;
initial begin
        for (i=0;i<99;i=i+1)begin
A_tb=$random;
B_tb=$random;
C_tb=$random;
D_tb=$random;
Sel_tb=$random;
Out_excpected=(Sel_tb==1)?(~(A_tb^B_tb^C_tb)):((D_tb[0]&D_tb[1])|(D_tb[2]));
Out_bar_excpected =~(Out_excpected);
#10
if(Out_excpected!=Out_dut)begin
$display("Error...");
$stop;
end
        end
```

```
$stop;
end
initial begin
$monitor("Out_excpected =%d , Out_dut=%d",Out_excpected,Out_dut);
end
endmodule
```





[Q2] 2) Design a 4-bit priority encoder, the following truth table is provided where x is 4-bit input and y is a 2-bit output. Randomize the stimulus in the testbench and add an expected result y in your testbench code and make the testbench self-checking.

x3	x2	x1	x0	y1	y0
1	X	X	X	1	1
0	1	X	X	1	0
0	0	1	X	0	1
0	0	0	X	0	0

The design code:

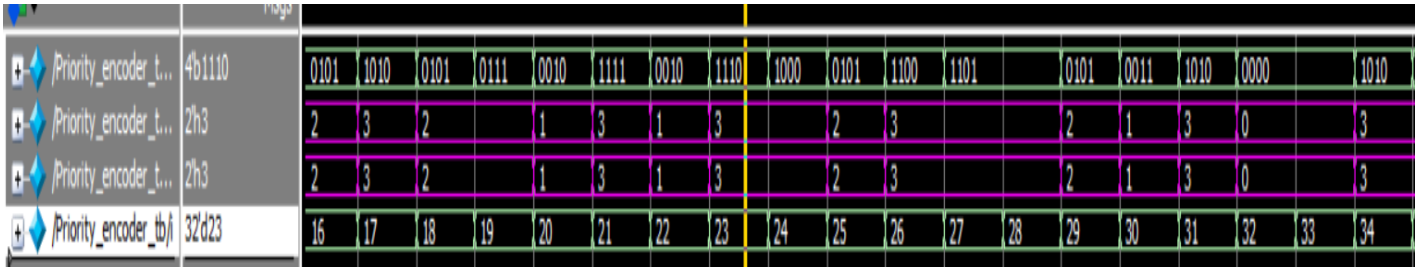
```
module Priority_encoder(X,Y);
input [3:0] X;
output reg [1:0] Y ;
always @(*) begin
if (X[3]==1)
Y=3;
else if (X[2]==1)
Y=2;
else if (X[1]==1)
Y=1;
else
Y=0;
end
endmodule
```

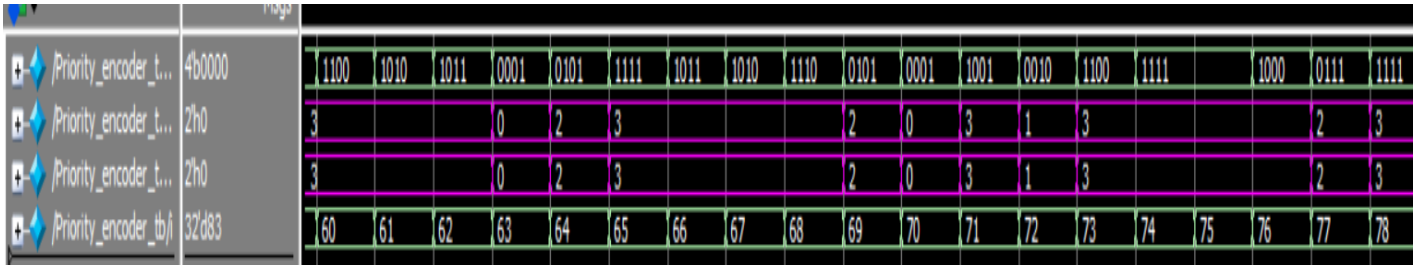
The testbench code:

```
module Priority_encoder_tb();
reg [3:0] X_tb;
reg [1:0] Y_excpected;
wire [1:0] Y_dut;
Priority_encoder dut(X_tb,Y_dut);
integer i;
initial begin
        for(i=0;i<99;i=i+1)begin
X_tb=$random;
casex(X_tb)
'b1xxx:Y_excpected=3;
'b01xx:Y_excpected=2;
'b001x:Y_excpected=1;
'b000x:Y_excpected=0;
endcase
#10
if(Y_excpected!=Y_dut)begin
$display("Error....");
$stop;
end
```

```
        end
$stop;
end
initial begin
$monitor ("Y_excpected=%d , Y_dut=%d",Y_excpected,Y_dut);
end
endmodule
```

Note: I do not need to use else or default because I wrote all cases





[Q3]

The design code:

```
module BCD(D,Y);
input [9:0] D;
output reg [3:0] Y;
always @(*) begin
  case(D)
2:Y=1;
4:Y=2;
8:Y=3;
16:Y=4;
32:Y=5;
64:Y=6;
128:Y=7;
256:Y=8;
512:Y=9;
default: Y=0;
  endcase
end
endmodule
```

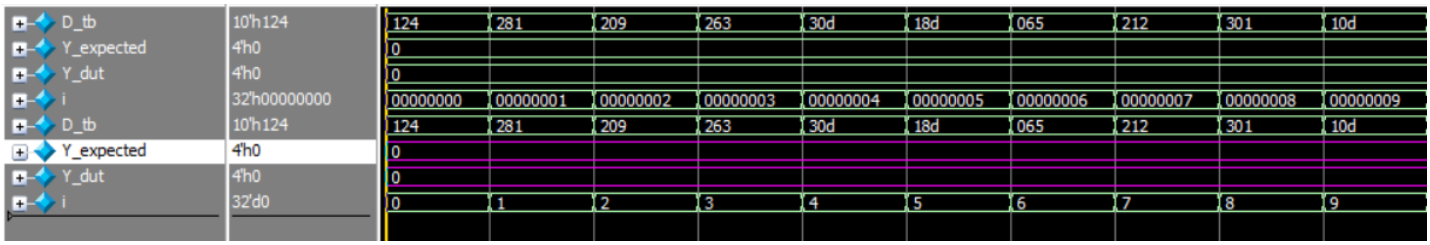
3) Design a decimal to BCD "Binary Coded Decimal" encoder has 10 input lines D0 to D9 and 4 output lines Y0 to Y3 . Below is the truth table for a decimal to BCD encoder. Output should be held LOW if none of the following input patterns is observed. Randomize the stimulus in the testbench and add an expected result y in your testbench code and make the testbench self-checking.

Input										Output			
D_9	D_8	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	1	0
0	0	0	0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0	0	0	0	1	0	1
0	0	0	1	0	0	0	0	0	0	0	1	1	0
0	0	1	0	0	0	0	0	0	0	0	1	1	1
0	1	0	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0	0	0	1	0	0	1

The testbench code:

```
module BCD_tb();
reg [9:0] D_tb;
reg [3:0] Y_expected;
wire [3:0] Y_dut;
BCD dut(D_tb,Y_dut);
integer i;
initial begin
        for(i=0;i<1000;i=i+1)begin
D_tb=$random;
if (D_tb==2)Y_expected=1;
else if(D_tb==4)Y_expected=2;
else if(D_tb==8)Y_expected=3;
else if(D_tb==16)Y_expected=4;
else if(D_tb==32)Y_expected=5;
else if(D_tb==64)Y_expected=6;
else if(D_tb==128)Y_expected=7;
```

```
else if(D_tb==256)Y_expected=8;
else if(D_tb==512)Y_expected=9;
else Y_expected=0;
#10
if (Y_expected!=Y_dut) begin
$display("ERROR...");
$stop;
end
        end
$stop;
end
initial begin
$monitor("Y_expected=%d , Y_dut=%d , iteration=%d",Y_expected,Y_dut,i);
end
endmodule
```



4) Implement N-bit adder using Dataflow modeling style

- The design takes 2 inputs (A, B) and the summation is assigned to output (C) ignoring the carry
- Parameter N has default value = 1.
- Randomize the stimulus in the testbench and add an expected result y in your testbench code and make the testbench self-checking.

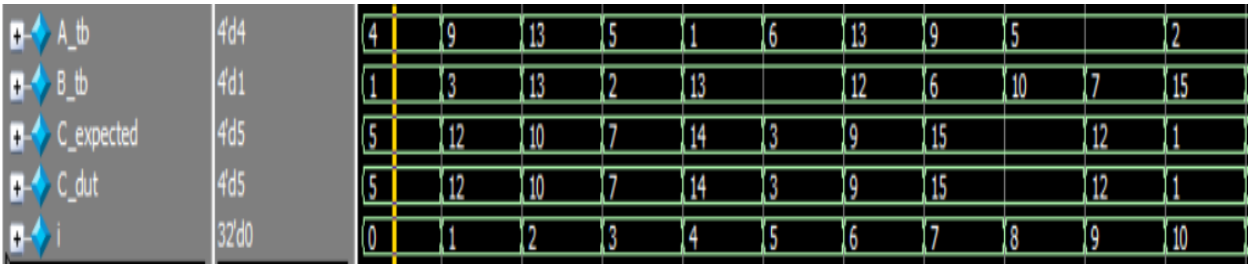
[Q4]

The design code:

```
module N_bit_Adder(A,B,C);
parameter N = 1;
input [N-1:0] A,B;
output [N-1:0] C ;
assign C = A+B;
endmodule
```

The testbench code:

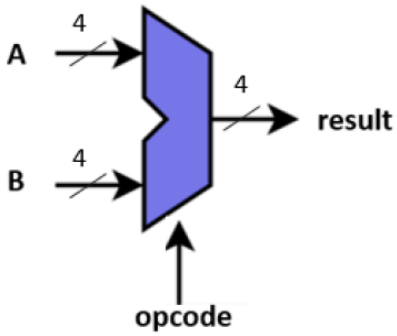
```
module N_bit_Adder_tb();
parameter N_tb =4 ;
reg [N_tb-1:0] A_tb,B_tb,C_expected;
wire[N_tb-1:0] C_dut;
N_bit_Adder #(N_tb) dut(A_tb,B_tb,C_dut);
integer i ;
initial begin
        for(i=0;i<99;i=i+1)begin
A_tb=$random;
B_tb=$random;
C_expected=A_tb+B_tb;
#10
if(C_dut!=C_expected)begin
$display("Errror...");
$stop;
end
        end
$stop;
end
initial begin
$monitor("A_tb=%d , B_tb=%d , C_expected=%d ,iteration=%d",A_tb,B_tb,C_expected,i);
end
endmodule
```



5) Design N-bit ALU that perform the following operations

- The design has 3 inputs and 1 output
- Instantiate the half adder from the previous design to implement the addition operation of the ALU
- For the subtraction, subtract B from A "A – B"
- Parameter N has default value = 4.
- Randomize the stimulus in the testbench and add an expected result y in your testbench code and make the testbench self-checking.

Inputs		Outputs
opcode		Operation
0	0	Addition
1	0	Subtraction
0	1	OR
1	1	XOR



A diagram of a diag
diagram

Description automa

[Q5]

The design code:

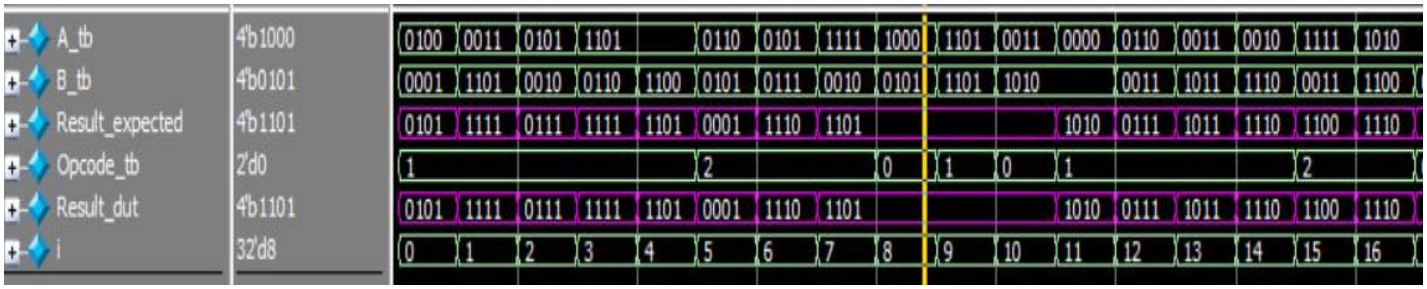
```
module N_bit_Adder(A,B,C);
parameter N = 1;
input [N-1:0] A,B;
output [N-1:0] C ;
assign C = A+B;
endmodule
```

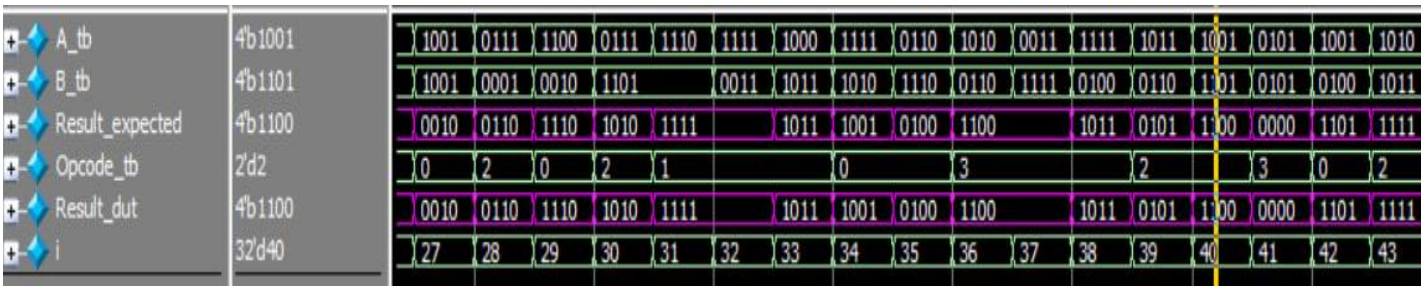
```
module  N_bit_Alu(A,B,Opcode,Result);
parameter N = 4 ;
input [N-1:0] A,B;
input [1:0] Opcode ;
output reg [N-1:0] Result ;
wire [N-1:0] Adder_result ;
N_bit_Adder #(N) m1(A,B,Adder_result);
always @(*) begin
        if (Opcode==0)
Result=Adder_result;
        else if (Opcode==1)
Result=A|B;
  else if (Opcode==2)
Result=A-B;
  else if (Opcode==3)
Result=A^B;
end
endmodule
```

The testbench code:

```
module N_bit_Alu_tb();
parameter N_tb =4;
reg [N_tb-1:0] A_tb,B_tb,Result_expected;
reg [1:0] Opcode_tb;
wire [N_tb-1:0] Result_dut;
N_bit_Alu #(N_tb) dut(A_tb,B_tb,Opcode_tb,Result_dut);
integer i ;
initial begin
for(i=0;i<99;i=i+1)begin
```

```
A_tb=$random;
B_tb=$random;
Opcode_tb=$random;
        case(Opcode_tb)
0:Result_expected=A_tb+B_tb;
1:Result_expected=A_tb|B_tb;
2:Result_expected=A_tb-B_tb;
3:Result_expected=A_tb^B_tb;
endcase
#10
if(Result_expected!=Result_dut)begin
$display ("Errror...");
$stop;
end
end
$stop;
end
initial begin
$monitor("A_tb=%d , B_tb=%d , Opcode_tb=%d ,
Result_expected=%d",A_tb,B_tb,Opcode_tb,Result_expected);
end
endmodule
```



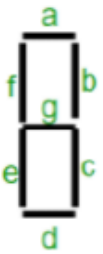


[Q6]

6) Implement 4-bit ALU display on 7 Segment LED Display

- The design has 4 inputs: A, B, opcode, enable.
- The design has 7 outputs (a-g)
- Instantiate the N-bit ALU designed in the previous design with parameter N = 4
- ALU should execute the operation on A and B depending on the input opcode
- ALU output should be considered as the digit to be displayed on the 7 segment LED display
- Below the truth table of the 7-segment decoder
- Since we have verified the ALU and Adder, we need to make sure that the outputs a to g are correct. Write 16 directed test vectors to exercise all the below digits with enable input equals to 1 then one directed test vector to drive the enable to 0. Make the testbench self-checking.

	Input	Output						
Digit	enable	a	b	c	d	e	f	g
0	1	1	1	1	1	1	1	0
1	1	0	1	1	0	0	0	0
2	1	1	1	0	1	1	0	1
3	1	1	1	1	1	0	0	1
4	1	0	1	1	0	0	1	1
5	1	1	0	1	1	0	1	1
6	1	1	0	1	1	1	1	1
7	1	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1	1
9	1	1	1	1	1	0	1	1
A	1	1	1	1	0	1	1	1
b	1	0	0	1	1	1	1	1
C	1	1	0	0	1	1	1	0
d	1	0	1	1	1	1	0	1
E	1	1	0	0	1	1	1	1
F	1	1	0	0	0	1	1	1
x	0	0	0	0	0	0	0	0



The design code:

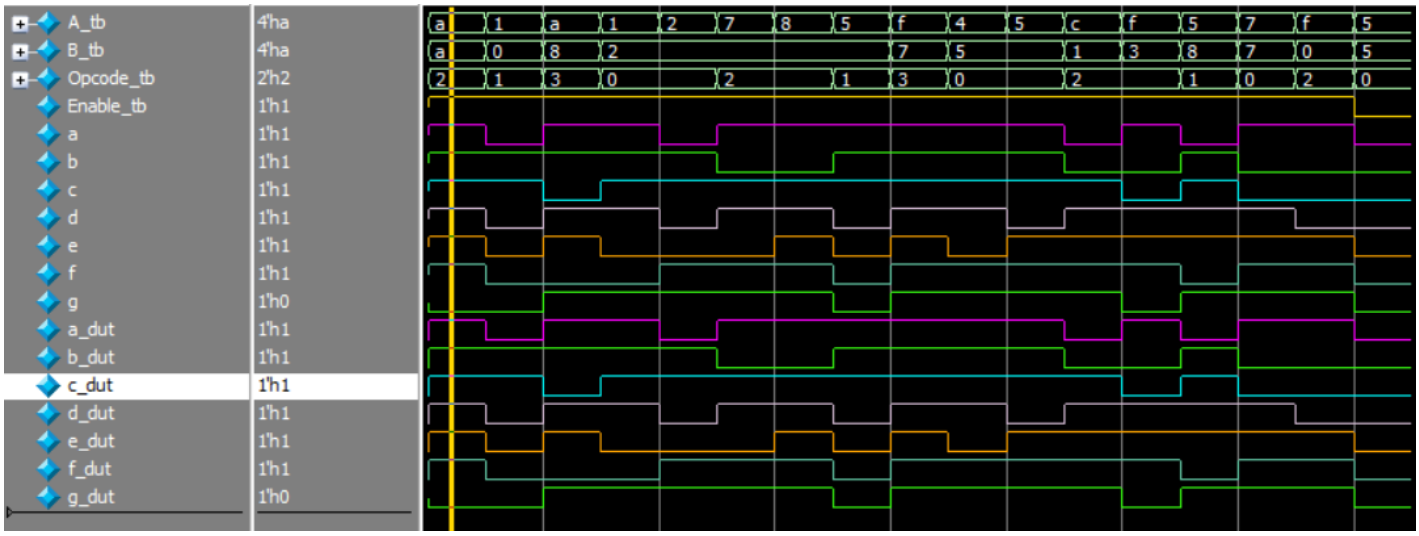
```
module N_bit_Adder(A,B,C);
parameter N = 1;
input [N-1:0] A,B;
output [N-1:0] C ;
assign C = A+B;
endmodule


module  N_bit_Alu(A,B,Opcode,Result);
parameter N = 4 ;
input [N-1:0] A,B;
input [1:0] Opcode ;
output reg [N-1:0] Result ;
wire [N-1:0] Adder_result ;
N_bit_Adder #(N) m1(A,B,Adder_result);
always @(*) begin
        if (Opcode==0)
Result=Adder_result;
        else if (Opcode==1)
Result=A|B;
  else if (Opcode==2)
Result=A-B;
  else if (Opcode==3)
Result=A^B;
end
endmodule

module SSD(A,B,Opcode,Enable,a,b,c,d,e,f,g);
parameter N_SSD = 4;
input [N_SSD-1:0] A,B;
input [1:0] Opcode;
input Enable ;
output reg a,b,c,d,e,f,g;
wire [N_SSD-1:0] Result_alu;
N_bit_Alu #(N_SSD) Alu(A,B,Opcode,Result_alu);
always @(*) begin
        case({Result_alu,Enable})
5'b00001:{a,b,c,d,e,f,g}=7'b1111110;//0
5'b00011:{a,b,c,d,e,f,g}=7'b0110000;//1
```

```
5'b00101:{a,b,c,d,e,f,g}=7'b1101101;//2
5'b00111:{a,b,c,d,e,f,g}=7'b1111001;//3
5'b01001:{a,b,c,d,e,f,g}=7'b0110011;//4
5'b01011:{a,b,c,d,e,f,g}=7'b1011011;//5
5'b01101:{a,b,c,d,e,f,g}=7'b1011111;//6
5'b01111:{a,b,c,d,e,f,g}=7'b1110000;//7
5'b10001:{a,b,c,d,e,f,g}=7'b1111111;//8
5'b10011:{a,b,c,d,e,f,g}=7'b1111011;//9
5'b10101:{a,b,c,d,e,f,g}=7'b1110111;//a
5'b10111:{a,b,c,d,e,f,g}=7'b0011111;//b
5'b11001:{a,b,c,d,e,f,g}=7'b1001110;//c
5'b11011:{a,b,c,d,e,f,g}=7'b0111101;//d
5'b11101:{a,b,c,d,e,f,g}=7'b1001111;//e
5'b11111:{a,b,c,d,e,f,g}=7'b1000111;//f
default : {a,b,c,d,e,f,g}=0;
endcase
end
endmodule
```

The testbench code:

```
module SSD_tb();
parameter N_SSD_tb = 4;
reg [N_SSD_tb-1:0] A_tb,B_tb;
reg [1:0] Opcode_tb;
reg Enable_tb,a,b,c,d,e,f,g; //a:g expected
wire a_dut,b_dut,c_dut,d_dut,e_dut,f_dut,g_dut;
SSD #(N_SSD_tb) S0(A_tb,B_tb,Opcode_tb,Enable_tb,
                   a_dut,b_dut,c_dut,d_dut,e_dut,f_dut,g_dut);
//Note OP=0:ADD,OP=1:OR,OP=2:SUB,OP=3:XOR
initial begin
#0 A_tb=10;B_tb=10;Opcode_tb=2;Enable_tb=1;{a,b,c,d,e,f,g}=7'b1111110;//0
#10 A_tb=1;B_tb=0;Opcode_tb=1;Enable_tb=1;{a,b,c,d,e,f,g}=7'b0110000;//1
#10 A_tb=10;B_tb=8;Opcode_tb=3;Enable_tb=1;{a,b,c,d,e,f,g}=7'b1101101;//2
#10 A_tb=1;B_tb=2;Opcode_tb=0;Enable_tb=1;{a,b,c,d,e,f,g}=7'b1111001;//3
#10 A_tb=2;B_tb=2;Opcode_tb=0;Enable_tb=1;{a,b,c,d,e,f,g}=7'b0110011;//4
#10 A_tb=7;B_tb=2;Opcode_tb=2;Enable_tb=1;{a,b,c,d,e,f,g}=7'b1011011;//5
#10 A_tb=8;B_tb=2;Opcode_tb=2;Enable_tb=1;{a,b,c,d,e,f,g}=7'b1011111;//6
#10 A_tb=5;B_tb=2;Opcode_tb=1;Enable_tb=1;{a,b,c,d,e,f,g}=7'b1110000;//7
#10 A_tb=15;B_tb=7;Opcode_tb=3;Enable_tb=1;{a,b,c,d,e,f,g}=7'b1111111;//8
```

```
#10 A_tb=4;B_tb=5;Opcode_tb=0;Enable_tb=1;{a,b,c,d,e,f,g}=7'b1111011;//9
#10 A_tb=5;B_tb=5;Opcode_tb=0;Enable_tb=1;{a,b,c,d,e,f,g}=7'b1110111;//a
#10 A_tb=12;B_tb=1;Opcode_tb=2;Enable_tb=1;{a,b,c,d,e,f,g}=7'b0011111;//b
#10 A_tb=15;B_tb=3;Opcode_tb=2;Enable_tb=1;{a,b,c,d,e,f,g}=7'b1001110;//c
#10 A_tb=5;B_tb=8;Opcode_tb=1;Enable_tb=1;{a,b,c,d,e,f,g}=7'b0111101;//d
#10 A_tb=7;B_tb=7;Opcode_tb=0;Enable_tb=1;{a,b,c,d,e,f,g}=7'b1001111;//e
#10 A_tb=15;B_tb=0;Opcode_tb=2;Enable_tb=1;{a,b,c,d,e,f,g}=7'b1000111;//f
#10 A_tb=5;B_tb=5;Opcode_tb=0;Enable_tb=0;{a,b,c,d,e,f,g}=0;
#10
if({a_dut,b_dut,c_dut,d_dut,e_dut,f_dut,g_dut}!={a,b,c,d,e,f,g})begin
$display("Errrrror...");
$stop;
end
$stop;
end
initial begin
$monitor("A_tb=%d,B_tb=%d,Opcode_tb=%d,Enable_tb=%d,{a,b,c,d,e,f,g}=%b",
        A_tb,B_tb,Opcode_tb,Enable_tb,{a,b,c,d,e,f,g});
end
endmodule
```




```
# A_tb=10,B_tb=10,Opcode_tb=2,Enable_tb=1,{a,b,c,d,e,f,g}=1111110
# A_tb= 1,B_tb= 0,Opcode_tb=1,Enable_tb=1,{a,b,c,d,e,f,g}=0110000
# A_tb=10,B_tb= 8,Opcode_tb=3,Enable_tb=1,{a,b,c,d,e,f,g}=1101101
# A_tb= 1,B_tb= 2,Opcode_tb=0,Enable_tb=1,{a,b,c,d,e,f,g}=1111001
# A_tb= 2,B_tb= 2,Opcode_tb=0,Enable_tb=1,{a,b,c,d,e,f,g}=0110011
# A_tb= 7,B_tb= 2,Opcode_tb=2,Enable_tb=1,{a,b,c,d,e,f,g}=1011011
# A_tb= 8,B_tb= 2,Opcode_tb=2,Enable_tb=1,{a,b,c,d,e,f,g}=1011111
# A_tb= 5,B_tb= 2,Opcode_tb=1,Enable_tb=1,{a,b,c,d,e,f,g}=1110000
# A_tb=15,B_tb= 7,Opcode_tb=3,Enable_tb=1,{a,b,c,d,e,f,g}=1111111
# A_tb= 4,B_tb= 5,Opcode_tb=0,Enable_tb=1,{a,b,c,d,e,f,g}=1111011
# A_tb= 5,B_tb= 5,Opcode_tb=0,Enable_tb=1,{a,b,c,d,e,f,g}=1110111
# A_tb=12,B_tb= 1,Opcode_tb=2,Enable_tb=1,{a,b,c,d,e,f,g}=0011111
# A_tb=15,B_tb= 3,Opcode_tb=2,Enable_tb=1,{a,b,c,d,e,f,g}=1001110
# A_tb= 5,B_tb= 8,Opcode_tb=1,Enable_tb=1,{a,b,c,d,e,f,g}=0111101
# A_tb= 7,B_tb= 7,Opcode_tb=0,Enable_tb=1,{a,b,c,d,e,f,g}=1001111
# A_tb=15,B_tb= 0,Opcode_tb=2,Enable_tb=1,{a,b,c,d,e,f,g}=1000111
# A_tb= 5,B_tb= 5,Opcode_tb=0,Enable_tb=0,{a,b,c,d,e,f,g}=0000000
```